

Programación

Array y ArrayList

Unidad 8

Jesús Alberto Martínez
versión 0.2



Reconocimiento – NoComercial – CompartirIgual (by-nc-sa): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.
Basado en los apuntes del CEEDCV



Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:



Importante



Atención

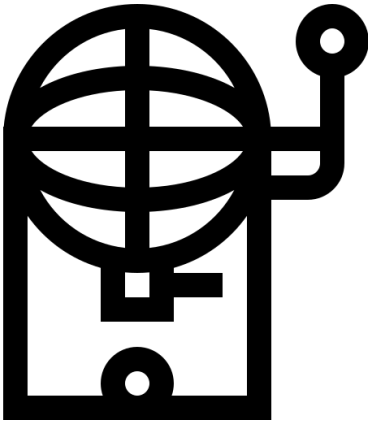


Interesante

Unidad 8. Array y ArrayList

0 Problema.....	3
1 Introducción.....	4
2 Propiedades.....	4
3 Vectores (Arrays unidimensionales).....	5
3.1 Declaración.....	5
3.2 Instancia.....	5
3.3 Almacenamiento.....	6
3.4 Longitud de un vector.....	6
3.5 Recorrido de un vector.....	7
3.6 Copia de vectores.....	8
4 Arrays multidimensionales.....	9
5 Foreach, BUCLE FOR MEJORADO.....	10
6 La clase Arrays (api completa).....	10
7 Búsqueda en Vectores.....	12
7.1 Búsqueda secuencial.....	12
7.2 Búsqueda dicotómica o binaria.....	12
8 Ordenación de vectores.....	13
9 Ejemplo de llenado y recorrido de un vector.....	14
10 Ejecución de programas Java y paso de parámetros.....	15
11 La clase ArrayList.....	18
11.1 Introducción.....	18
11.2 Declaración.....	18
11.3 Llenado.....	18
11.4 Métodos de Acceso y Manipulación.....	19
11.5 Recorrido de un ArrayList.....	20
11.6 Ejemplo 1.....	21
11.7 Ejemplo 2.....	22
11.8 Ampliación.....	24
Definición.....	24
Clases envoltorio o wrapper.....	24
Recorrido.....	24
Copia de un ArrayList.....	25
ArrayList Bidimensionales.....	25
Clase Collections.....	25
11.9 Ejercicios.....	26

0 Problema



Pensemos en el siguiente ejercicio.

Queremos generar automáticamente una combinación para la lotería primitiva., son 6 números al azar.

¿Cómo lo podemos solucionar?

6 variables, una para cada número, y sacamos un número entre 1 y 49 para cada una

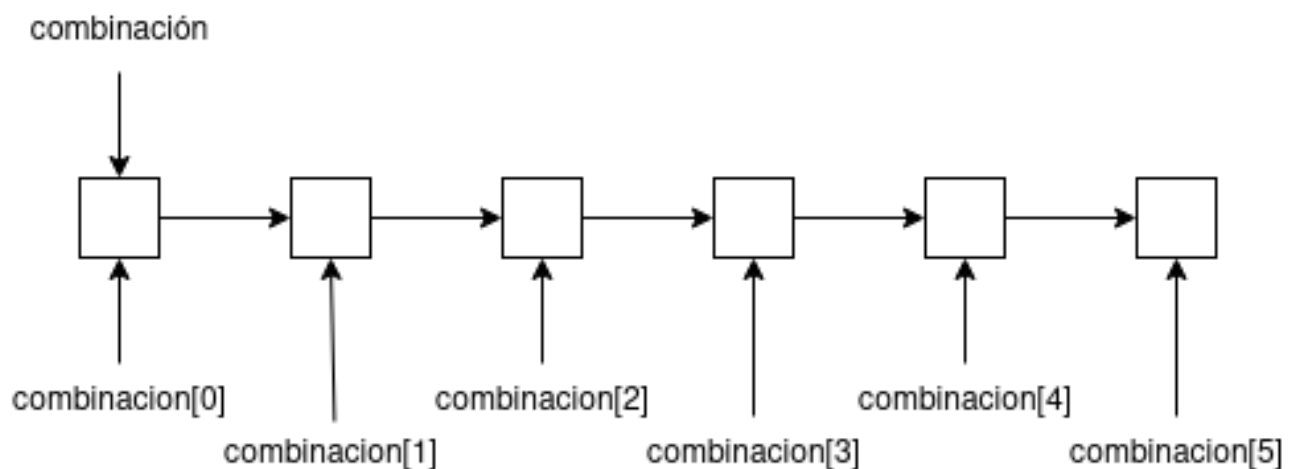
Solucionado.

Pero, ¿y si queremos comprobar que no ha salido un número repetido?

Sacamos el primero, sacamos el segundo y comprobamos que no sea como el primero, sacamos el tercero y comprobamos que no sea como el primero no como el segundo, y así hasta llegar al sexto. Solucionado, pero, ¿no podemos hacerlo más fácil?

Ahora, en vez de sacar los 6 primeros números al azar, quiero sacar 1000000 de números entre el 1 y el 49, y quedarme con los 6 que más veces han salido. ¿Tenemos solución?

```
int[] combinacion=new int[6];
```



1 Introducción



Un **array** o **vector** es una colección de valores de un **mismo tipo** dentro de una misma variable. De forma que se puede acceder a cada valor independientemente.

Para Java, además, un array es un objeto que tiene propiedades que se pueden manipular.

Los arrays solucionan problemas concernientes al manejo de muchas variables que se refieren a datos similares.

Por ejemplo si tuviéramos la necesidad de almacenar las notas de una clase con 18 alumnos, necesitaríamos 18 variables, con la tremenda lentitud de manejo que supone eso. Solamente calcular la nota media requeriría una tremenda línea de código. Almacenar las notas supondría al menos 18 líneas de código.

En lugar de crear 18 variables sería mucho mejor crear un array de tamaño 18 (es como si tuviéramos una sola variable que puede almacenar varios valores).

Gracias a los arrays se puede crear un conjunto de variables con el mismo nombre. La diferencia será que un número (índice del array) distinguirá a cada variable.

2 Propiedades

Algunas propiedades de los arrays son:

- Los arrays se utilizan como contenedores para almacenar datos relacionados (en lugar de declarar variables por separado para cada uno de los elementos del array).
- **Todos los datos incluidos en el array son del mismo tipo.** Se pueden crear arrays de enteros de tipo *int* o de reales de tipo *float*, pero **en un mismo array no se pueden mezclar tipos de datos**, por ej. *int* y *float*.
- El tamaño del array se establece cuando se crea el array (con el operador *new*, igual que cualquier otro objeto).
- A los elementos del array se accederá a través de la posición que ocupan dentro del conjunto de elementos del array.
- Los arrays unidimensionales se conocen con el nombre de **vectores**.
- Los arrays bidimensionales se conocen con el nombre de **matrices**.

3 Vectores (Arrays unidimensionales)

3.1 Declaración

Un array se declara de forma similar a una variable simple pero añadiendo corchetes para indicar que se trata de un array y no de una variable simple del tipo especificado.

Un Vector (array unidimensional) se puede declarar de dos formas:

- tipo identificador[];
- tipo[] identificador;

Donde tipo es el tipo de dato de los elementos del vector e identificador es el nombre de la variable.

Ejemplos:

```
int notas[];  
double cuentas[];
```

Declara un array de tipo int y otro de tipo double. Esta declaración indica para qué servirá el array, pero no reserva espacio en la memoria RAM al no saberse todavía el tamaño del mismo. Todavía no puede utilizarse el array, falta instanciarlo.

3.2 Instancia

Tras la declaración del array, se tiene que instanciar, para ello se utiliza el operador **new**, que es el que realmente crea el array indicando un tamaño. Cuando se usa **new**, es cuando se reserva el espacio necesario en memoria. Un array no inicializado es un array **null** (sin valor).

Ejemplo:

```
int notas[]; // Declaramos 'notas' como array de tipo int  
notas = new int[3]; // Instanciamos 'notas' a tamaño 3  
  
// Es habitual declarar e instanciar en una sola línea  
int notas[] = new int[3];
```

En el ejemplo anterior se crea un array de tres enteros (con los tipos básicos se crea en memoria el array y se inicializan los valores, los números se inicializan a 0).

3.3 Almacenamiento

Los valores del array se asignan (almacenan) utilizando el índice del mismo entre corchetes.

⚡ El **primer elemento del vector** siempre estará en la posición o **índice 0**

Índices →	0	1	2	3	4
Valores →	8	10	2	3	5

Por ejemplo, para almacenar el valor 8 en la tercera posición del array escribiríamos:

```
notas[2] = 2;
```

También se pueden asignar valores al array en la propia declaración e instanciación:

```
int notas[] = {8, 10, 2, 3, 5};
int notas2[] = new int[] {8, 10, 2, 3, 5}; //Equivalente a la anterior
```

Esto declara e inicializa un array de tres elementos. El ejemplo sería equivalente a:

```
notas[0] = 8;
notas[1] = 10;
notas[2] = 2;
notas[3] = 3;
notas[4] = 5;
```

En Java (como en otros lenguajes) el primer elemento de un array está en la posición **cero**. El primer elemento del array notas, es notas[0].

📦 Se pueden declarar arrays a cualquier tipo de datos (enteros, booleanos, doubles, ... e incluso objetos).

3.4 Longitud de un vector

Los arrays poseen una propiedad llamada length que indica su tamaño.

Ejemplo:

```
int notas[] = new int[4]; // Declara e instancia vector tipo int de tamaño 4
System.out.println( notas.length ); // Mostrará un 4
```

Si el vector tiene como en el ejemplo 4 elementos, la propiedad length nos devolverá el valor entero 4, pero su primer elemento se encuentra en notas[0] y el último en notas[3].

3.5 Recorrido de un vector

Para recorrer un vector (acceder a todos sus elementos) siempre será necesario un bucle.

En el siguiente ejemplo declaramos e instanciamos un vector tipo `int` con las notas de un alumno y luego utilizamos un bucle `for` para recorrer el vector y mostrar todos los elementos.

```
// Declaramos e instanciamos vector tipo int
int notas[] = new int[] {7, 3, 9, 6, 5};

// Como el vector es de tamaño 5 sus elementos estarán en las posiciones de 0
// a 4
// Recorremos el vector desde i=0 hasta i<5 (es decir, desde 0 hasta 4)
for (int i = 0; i < notas.length; i++) {
    System.out.println(notas[i]);
}
```

Ahora vamos a calcular la nota media (sumar todas y luego dividir entre el número de notas):

```
// Declaramos suma y media
int suma = 0;
int media;

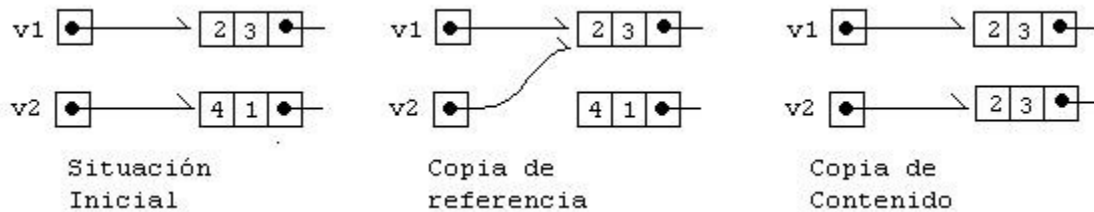
// Recorremos el vector desde 0 hasta 4, acumulando las notas en suma
for (int i = 0; i < notas.length; i++) {
    suma += notas[i];    // Equivale a: suma = suma + notas[i]
}

// Calculamos la media y la mostramos por pantalla
media = suma / notas.length;
System.out.println("La nota media es: " + media);
```

3.6 Copia de vectores

Para copiar vectores no basta con igualar un vector a otro como si fuera una variable simple.

Si partimos de dos vectores v1, v2 e hiciéramos v2=v1, lo que ocurriría sería que v2 apuntaría a la posición de memoria de v1. Eso es lo que se denomina un copia de referencia:



Si por ejemplo queremos copiar todos los elementos del vector v2 en el vector v1, existen dos formas para hacerlo:

- **Copiar los elementos uno a uno**

```
for (i = 0; i < v1.length; i++)
    v2[i] = v1[i];
```

- **Utilizar la función arraycopy**

```
System.arraycopy(v_origen, i_origen, v_destino, i_destino, length);
```

v_origen: Vector origen

i_origen: Posición inicial de la copia

v_destino: Vector destino

i_destino: Posición final de la copia

length: Cantidad de elementos a copiar

```
// Copiamos todos los elementos de v1 en v2
System.arraycopy(v1, 0, v2, 0, v1.length);
```


4 Arrays multidimensionales

Los arrays pueden tener más de una dimensión. Los más utilizados son los arrays de 2 dimensiones, conocidos como matrices.

 Las matrices se definen de las siguientes formas:

```
tipo identificador[] [];  
tipo[] [] identificador;
```

Por ejemplo, declaramos e instanciamos un array de 3 x 3 (3 filas x 3 columnas).

```
double precios[] [] = new int[3][3];
```

Accedemos a sus valores utilizando dobles corchetes.

```
precios[0][0] = 7.5;  
precios[0][1] = 12;  
precios[0][2] = 0.99;  
precios[1][0] = 4.75;  
// etc.
```

		Columnas		
		0	1	2
Filas	0	(0,0)	(0,1)	(0,2)
	1	(1,0)	(1,1)	(1,2)
	2	(2,0)	(2,1)	(2,2)

Veamos otro ejemplo en el que declaramos e instanciamos un array de 3 filas x 6 columnas para almacenar las notas de 3 alumnos (la fila corresponde a un alumno, y cada columna a las notas de dicho alumno):

```
int notas[] [] = new int[3][6]; // Es equivalente a 3 vectores de tamaño 6
```

Suponiendo que las notas ya están almacenadas, vamos a recorrer la matriz (array bidimensional) para mostrar las notas por pantalla. Hay que tener en cuenta que como tiene dos dimensiones necesitaremos un bucle anidado (uno para las filas y otro para las columnas de cada fila):

```
// Para cada fila (alumno)  
for (int i = 0; i < notas.length; i++) {  
    System.out.print("Notas del alumno " + i + ": ");  
    // Para cada columna (nota)  
    for (int j = 0; j < notas[i].length; j++) {  
        System.out.print(notas[i][j] + " ");  
    }  
}
```

5 Foreach, BUCLE FOR MEJORADO

A partir de Java 8 aparece lo que se puede llamar como bucle for mejorado, o bucle foreach. Está especialmente indicado para recorrer estructuras.

```
foreach, estructura

for (tipo_dato variableIteración: estructura){
    <instrucciones dentro del bucle>
}
```

Si tomamos el mismo ejemplo que en punto de recorrido de un vector, podríamos realizarlo con el bucle for mejorado de esta forma:

```
// Declaramos e instanciamos vector tipo int
int notas[] = new int[] {7, 3, 9, 6, 5};

// Recorremos todos los elementos del array
for (int cadaNota : notas) {
    System.out.println(cadaNota);
}
```

en cada iteración del bucle, el valor de cada elemento del array lo tenemos en la variable de iteración.

Importante, no tenemos índice, si necesitamos el índice para algo tendremos que usar el for normal. Se usa esta estructura cuando hay que recorrer una estructura y no es necesario el índice.

6 La clase Arrays ([api completa](#))

En el paquete java.util se encuentra una clase estática llamada Arrays. Esta clase estática permite ser utilizada como si fuera un objeto (como ocurre con Math). Esta clase posee métodos muy interesantes para utilizar sobre arrays.

Su uso es:

```
Arrays.método(argumentos);
```

Algunos métodos son:

- **fill:** permite rellenar todo un array unidimensional con un determinado valor. Sus argumentos son el array a rellenar y el valor deseado:

Por ejemplo, llenar un array de 23 elementos enteros con el valor -1

```
int valores[] = new int[23];  
Arrays.fill(valores,-1); // Almacena -1 en todo el array 'valores'
```

También permite decidir desde qué índice hasta qué índice rellenamos:

```
Arrays.fill(valores,5,8,-1); // Almacena -1 desde el 5º a la 7º elemento
```

- **equals** : Compara dos arrays y devuelve true si son iguales (false en caso contrario). Se consideran iguales si son del mismo tipo, tamaño y contienen los mismos valores.

```
Arrays.equals(valoresA, valoresB); // devuelve true si los arrays son iguales
```

- **sort** : Permite ordenar un array en orden ascendente. Se pueden ordenar sólo una serie de elementos desde un determinado punto hasta un determinado punto.

```
int x[]={4,5,2,3,7,8,2,3,9,5};  
Arrays.sort(x); // Ordena x de menor a mayor  
Arrays.sort(x,2,5); // Ordena x solo desde 2º al 4º elemento
```

- **binarySearch** : Permite buscar un elemento de forma ultrarrápida en un array ordenado. Devuelve el índice en el que está colocado el elemento buscado. Ejemplo:

```
int x[]={1,2,3,4,5,6,7,8,9,10,11,12};  
Arrays.sort(x);  
System.out.println(Arrays.binarySearch(x,8)); //Devolvería 7
```

7 Búsqueda en Vectores

Existen dos formas de buscar un elemento dentro de un vector: la búsqueda secuencial y la búsqueda dicotómica o binaria.

7.1 Búsqueda secuencial

La búsqueda secuencial es la más fácil de las dos ya que consiste en comparar los elementos del vector con el elemento a buscar.

Un ejemplo es el siguiente, donde se devuelve la posición del elemento en el vector y si no lo encuentra, devuelve el valor -1:

```
56      public static int busquedaSecuencial(int[] v, int elemento)
57      {
58          int i, posicion = -1;
59
60          for(i = 0; i < v.length && posicion == -1; i++)
61              if(v[i] == elemento)
62                  posicion = i;
63
64          return posicion;
65      }
```

7.2 Búsqueda dicotómica o binaria

En este caso el vector debe estar ordenado. Se dividirá en dos para buscar el elemento en una parte del vector o en otra y así sucesivamente hasta encontrar, o no, el elemento.

Un ejemplo es el siguiente:

```
66 public static int busquedaDicotomica(int[] v, int elemento)
67 {
68     int izq = 0; // El índice 'izq' se establece en la posición 0
69     int der = v.length-1; // El índice 'der' se establece en la última posición
70     int centro = (izq + der)/2; // El índice 'centro' se establece en la posición central
71     int posicion;
72
73     while(izq <= der && v[centro] != elemento)
74     {
75         if(elemento < v[centro])
76             der = centro - 1; // Si el elemento es menor que el centro cambiamos el índice 'der'
77         else
78             izq = centro + 1; // Sino cambiamos el índice 'izq'
79
80         centro = (izq + der)/2; // Actualizamos el centro
81     }
82
83     if(izq > der)
84         posicion = -1;
85     else
86         posicion = centro;
87
88     return posicion;
89 }
```

En este caso, al igual que en el anterior, la función devuelve la posición del elemento o -1 si no lo encuentra. Es importante ver que el vector debe estar ordenado para quedarnos con la parte desde la izquierda al centro del vector o desde el centro del vector a la derecha, dependiendo si el elemento a buscar es mayor o menor al elemento del centro del vector.

Esta búsqueda es más óptima que la secuencial ya que no tiene que recorrer el vector entero.

8 Ordenación de vectores

Existen diferentes algoritmos para ordenar un vector. Algunos ejemplos son:

- Burbuja
- Inserción
- Selección
- Quicksort (el más rápido de los cuatro)

No es necesario saber cómo funcionan estos algoritmos ya que Java ya los tiene implementados y podemos utilizar métodos de ordenación muy fácilmente (por ejemplo con `Arrays.sort()`).

De todos modos es un tema fascinante. Quien tenga interés puede encontrar la explicación y el código en diferentes lenguajes en la siguiente página web:

https://en.wikibooks.org/wiki/Algorithm_Implementation/Sorting

9 Ejemplo de llenado y recorrido de un vector

Vamos a ver un ejemplo de cómo llenar y mostrar un vector de enteros:

```
19 public static void main(String[] args) {  
20     Scanner in = new Scanner(System.in);  
21  
22     int[] vector = new int[5]; // Creación de un vector de enteros de tamaño 5  
23     int i;  
24  
25     System.out.print("Introduce los valores del vector: ");  
26  
27     // Llenado del vector con valores desde teclado  
28     for(i = 0; i < vector.length; i++)  
29         vector[i] = in.nextInt();  
30  
31     System.out.print("El vector introducido es: ");  
32  
33     // Mostrar el vector  
34     for(i = 0; i < vector.length; i++)  
35         System.out.print(vector[i] + " ");  
36  
37     System.out.println();  
38 }  
39 }
```

Hacemos uso de la propiedad **length**. También podríamos haber puesto un 5 (tamaño del vector).

La salida es:

```
run:  
Introduce los valores del vector: 1 2 3 4 5  
El vector introducido es: 1 2 3 4 5  
BUILD SUCCESSFUL (total time: 4 seconds)
```

Podemos introducir los valores con espacios y una vez introducido el quinto número darle al 'intro'. O introducir un valor por línea.

10 Ejecución de programas Java y paso de parámetros

La ejecución de programas en Java se realiza mediante el comando **java**, que se encuentra en el Java Runtime Environment (JRE), y a la que hay que pasarle como parámetro el nombre de la clase que queremos ejecutar.

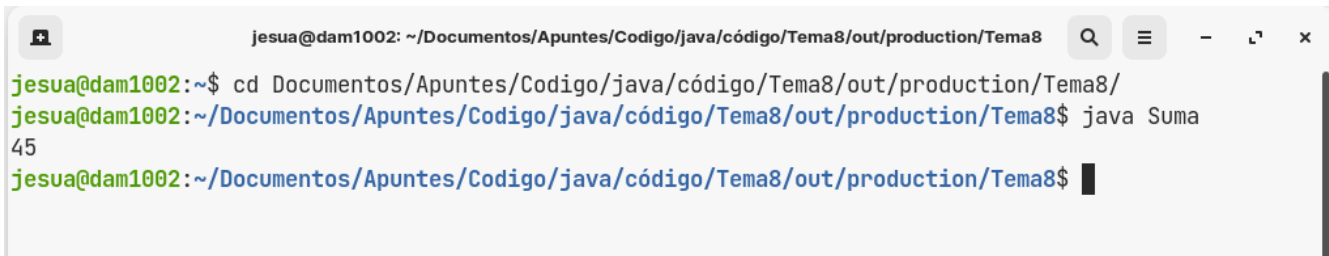
Imaginad que tenemos la siguiente clase

```
public class Suma {  
    public static void main(String[] args) {  
        int contador=0;  
        for (int i=0;i<10;i++)  
            contador+=i;  
        System.out.println(contador);  
    }  
}
```

Cuando ejecutamos una clase desde IntelliJ, realmente está llamando a la clase que queremos ejecutar con una serie de complicados parámetros y que previamente ha compilado con el comando **javac**, que se encuentra en el JDK.

```
/usr/lib/jvm/java-1.17.0-openjdk-amd64/bin/java  
-javaagent:/home/jesua/Aplicaciones/IntelliJ/lib/idea_rt.jar=34001:/home/  
jesua/Aplicaciones/IntelliJ/bin -Dfile.encoding=UTF-8 -classpath  
/home/jesua/Documentos/Apuntes/Codigo/java/código/Tema8/out/production/Tema8  
Suma
```

Pero nosotros podemos abrir un terminal, ir al directorio donde se encuentra la clase Suma y ejecutar simplemente



```
jesua@dam1002: ~/Documentos/Apuntes/Codigo/java/código/Tema8/out/production/Tema8  
jesua@dam1002:~$ cd Documentos/Apuntes/Codigo/java/código/Tema8/out/production/Tema8/  
jesua@dam1002:~/Documentos/Apuntes/Codigo/java/código/Tema8/out/production/Tema8$ java Suma  
45  
jesua@dam1002:~/Documentos/Apuntes/Codigo/java/código/Tema8/out/production/Tema8$
```

Ahora suponemos que la clase suma pertenece a un paquete, al paquete arrays por ejemplo. Entonces, para ejecutarla, iríamos al directorio base donde se encuentra el paquete, y en el comando compremos

```
java paquete.clase
```

En nuestro ejemplo

```
jesua@dam1002: ~/Documentos/Apuntes/Codigo/java/código/Tema8/out/production/Tema8
jesua@dam1002:~/Documentos/Apuntes/Codigo/java/código/Tema8/out/production/Tema8$ java arrays.Suma
45
jesua@dam1002:~/Documentos/Apuntes/Codigo/java/código/Tema8/out/production/Tema8$
```

Una vez que sabemos ejecutar un programa java, vamos a modificar su ejecución a través de la línea de comando. Modificamos nuestro programa de Suma, ahora queremos sumar una serie de cantidades que pasamos como parámetros a nuestra clase.

Queremos ejecutar

```
java arrays.Suma 13 90 1 65
```

y que nos muestre el resultado de sumar las cantidades que pasamos como parámetros.

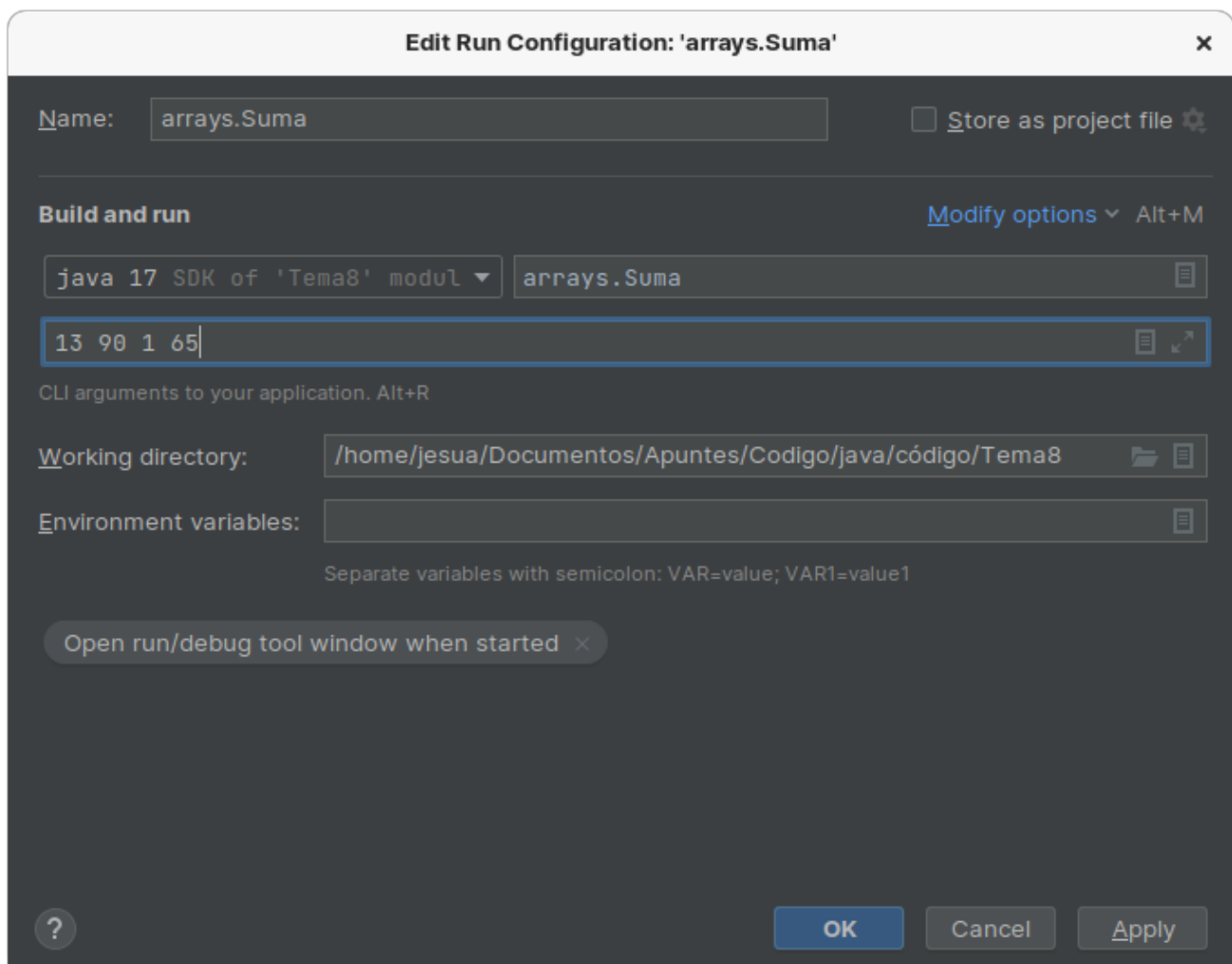
Para ello se utiliza el array de cadenas de caracteres que tiene siempre el método main. Él va a contener los parámetros que pasamos a la clase.

```
package arrays;

public class Suma {
    public static void main(String[] args) {
        int contador=0;
        if (args.length==0) {
            System.out.println("Debe introducir las cantidades a sumar");
            System.exit(1);
        }
        for (String numero : args) {
            if (numero.matches("\\d+"))
                contador+=Integer.parseInt(numero);
        }
        System.out.println(contador);
    }
}
```

```
jesua@dam1002: ~/Documentos/Apuntes/Codigo/java/código/Tema8/out/production/Tema8
jesua@dam1002:~/Documentos/Apuntes/Codigo/java/código/Tema8/out/production/Tema8$ java arrays.Suma
45
jesua@dam1002:~/Documentos/Apuntes/Codigo/java/código/Tema8/out/production/Tema8$ java arrays.Suma 13 90 1 65
169
```

Para pasar parámetros en la ejecución de IntelliJ hay que modificar los parámetros de ejecución de la clase (botón derecho... Modify Run Configuration...) y establecer los valores



11 La clase ArrayList

11.1 Introducción

Un *ArrayList* es una estructura de datos dinámica del tipo **colección** que implementa una **lista de tamaño variable**. Es similar a un *array* con las ventajas de que **su tamaño crece dinámicamente conforme se añaden elementos** (no es necesario fijar su tamaño al crearlo) y **permite almacenar datos y objetos de cualquier tipo**.

11.2 Declaración

Un *ArrayList* se declara como una clase más:

```
ArrayList lista = new ArrayList();
```

Necesitaremos importar la clase:

```
import java.util.ArrayList;
```

11.3 Llenado

Para insertar datos o elementos en un *ArrayList* puede utilizarse el método *add()*.

En el siguiente ejemplo insertamos datos *int*, *double*, *char* y *String*:

```
lista.add(-25);  
lista.add(3.14);  
lista.add('A');  
lista.add("Luis");
```

En el siguiente ejemplo insertamos objetos de la clase *Persona*:

```
lista.add(new Persona("28751533Q", "María", "Maida García", 37));  
lista.add(new Persona("65971552A", "Luis", "González Collado", 17));  
lista.add(new Persona("16834954R", "Raquel", "Dobón Pérez", 62));
```

Este código añade punteros a tres objetos del tipo *Persona* al *ArrayList*. Es decir, cada posición de la lista apuntará a un objeto *Persona* diferente. También se podría haber hecho lo siguiente:

```
Persona p = new Persona("28751533Q", "María", "Maida García", 37);  
  
lista.add(p);
```

11.4 Métodos de Acceso y Manipulación

Un *ArrayList* da a cada elemento insertado un índice numérico que hace referencia a la posición donde se encuentra, de forma similar a un array. Así, el primer elemento se encuentra almacenado en el índice 0, el segundo en el 1, el tercero en el 2 y así sucesivamente.

Los métodos más utilizados para acceder y manipular un *ArrayList* son:

- *int size()*; devuelve el número de elementos de la lista.
- *Elemento get(int index)*; devuelve una referencia al elemento en la posición index.
- *void clear()*; elimina todos los elementos de la lista. Establece el tamaño a cero.
- *boolean isEmpty()*; retorna true si la lista no contiene elementos.
- *boolean add(E element)*; inserta element al final de la lista y devuelve true.
- *void add(int index, E element)*; inserta element en la posición index de la lista. Desplaza una posición todos los demás elementos de la lista (no sustituye ni borra otros elementos).
- *void set(int index, E element)*; sustituye el elemento en la posición index por element.
- *boolean contains(Object o)*; busca el objeto o en la lista y devuelve true si existe. Utiliza el método *equals()* para comparar objetos.
- *int indexOf(Object o)*; busca el objeto o en la lista, empezando por el principio, y devuelve el índice dónde se encuentre. Devuelve -1 si no existe. Utiliza *equals()* para comparar objetos.
- *int lastIndexOf(Object o)*; como *indexOf()* pero busca desde el final de la lista.
- *E remove(int index)*; elimina el elemento en la posición index y lo devuelve.
- *boolean remove(Object obj)*; elimina la primera ocurrencia de obj en la lista. Devuelve true si lo ha encontrado y eliminado, false en otro caso. Utiliza *equals()* para comparar objetos.
- *void remove(int index)*; Elimina el objeto de la lista que se encuentra en la posición index. Es más rápido que el método anterior ya que no necesita recorrer toda la lista.

[Documentación oficial de ArrayList \(Java 17 API\)](#)

11.5 Recorrido de un ArrayList

Tenemos varias maneras diferentes en las que se puede iterar (recorrer) una colección del tipo ArrayList.

- Utilizando el bucle *for* y el método *get()* con un índice.

```
for(int i = 0; i < lista.size(); i++) {  
    System.out.println(lista.get(i)); // Lo imprimimos por pantalla  
}
```

- Utilizando un bucle *foreach*

```
for(Object objeto: lista) {  
    System.out.println(objeto); // Lo imprimimos por pantalla  
}
```

- Usando un objeto *Iterator* que permite recorrer listas como si fuese un índice. Se necesita importar la clase con *import java.util.Iterator;* Tiene dos métodos principales:

- *hasNext()*: Verifica si hay más elementos.
- *next()*: devuelve el objeto actual y avanza al siguiente.

```
Iterator iter = lista.iterator(); // Creamos el Iterator a partir de la lista  
while(iter.hasNext()) { // Mientras haya siguiente en la lista  
    System.out.println(iter.next()); // Lo imprimimos por pantalla  
}
```

11.6 Ejemplo 1

Ejemplo que crea, rellena y recorre un ArrayList de dos formas diferentes. Cabe destacar que, por defecto, el método `System.out.println()` invoca al método `toString()` de los elementos que se le pasen como argumento, por lo que realmente no es necesario utilizar `toString()` dentro de `println()`.

```
import java.util.ArrayList;
import java.util.Iterator;

public class Ejemplos {
    public static void main(String[] args) {
        // Creamos la lista
        ArrayList lista = new ArrayList();

        // Añadimos elementos al final de la lista
        lista.add("uno");
        lista.add("dos");
        lista.add("tres");
        lista.add("cuatro");

        // Añadimos un elemento en la posición 2
        lista.add(2, "dosdos");

        System.out.println(lista.size()); // Devuelve 5
        System.out.println(lista.get(0)); // Devuelve uno
        System.out.println(lista.get(1)); // Devuelve dos
        System.out.println(lista.get(2)); // Devuelve dosdos
        System.out.println(lista.get(3)); // Devuelve tres
        System.out.println(lista.get(4)); // Devuelve cuatro

        // Recorremos la lista y mostramos el contenido
        for (int i = 0; i < lista.size(); i++) {
            System.out.print(lista.get(i));
        }
        System.out.println();
        // Recorremos la lista con un Iterator
        // Creamos el iterator con un método de ArrayList
        Iterator it = lista.iterator();

        while (it.hasNext())
            System.out.print(it.next());
    }
}
```

Salida

```
5
uno
dos
dosdos
tres
cuatro
unodosdosdosdostrescuatro
unodosdosdosdostrescuatro
```

11.7 Ejemplo 2

Tenemos la clase Producto con:

- Dos atributos: nombre (String) y cantidad (int).
- Un constructor con parámetros.
- Un constructor sin parámetros.
- Métodos get y set asociados a los atributos.

```
public class Producto {  
    private String nombre;  
    private int cantidad;  
  
    // Constructores  
    public Producto(String nombre, int cantidad) {  
        this.nombre = nombre;  
        this.cantidad = cantidad;  
    }  
  
    public Producto() {  
        this.nombre = null;  
        this.cantidad=0;  
    }  
  
    // Métodos getters y setters  
    public String getNombre() {  
        return nombre;  
    }  
  
    public void setNombre(String nombre) {  
        this.nombre = nombre;  
    }  
  
    public int getCantidad() {  
        return cantidad;  
    }  
  
    public void setCantidad(int cantidad) {  
        this.cantidad = cantidad;  
    }  
}
```

En el programa principal creamos una lista de productos y realizamos operaciones sobre ella:

```
import java.util.ArrayList;

public class Ejemplo2 {
    public static void main(String[] args) {
        // Creamos los objetos
        Producto p1 = new Producto("Pan", 6);
        Producto p2 = new Producto("Leche", 2);
        Producto p3 = new Producto("Manzanas", 5);
        Producto p4 = new Producto("Brocoli", 2);
        Producto p5 = new Producto("Carne", 2);

        ArrayList lista = new ArrayList();

        // Insertamos los elementos en la lista
        lista.add(p1);
        lista.add(p2);
        lista.add(p3);
        lista.add(p4);

        lista.add(1, p5);

        lista.add(p5);

        // La lista tiene una referencia, modificamos el objeto
        Producto cambio = (Producto) lista.get(1);
        cambio.setCantidad(4);

        System.out.println("- Lista con " + lista.size() + " elementos");
        // Usamos el bucle foreach para recorrer todos los elementos
        for (Object o : lista) {
            Producto p = (Producto) o;
            System.out.println(p.getNombre() + " : " + p.getCantidad());
        }
        // Eliminamos un elemento
        lista.remove(2);
        System.out.println("- Lista con " + lista.size() + " elementos");
        for (Object o : lista) {
            Producto p = (Producto) o;
            System.out.println(p.getNombre() + " : " + p.getCantidad());
        }
        // Borramos todos los elementos de la lista
        lista.clear();
        System.out.println("- Lista con " + lista.size() + " elementos");
    }
}
```

11.8 Ampliación

Definición

A la hora de definir un ArrayList lo hemos realizado así:

```
ArrayList nombreArray = new ArrayList();
```

Aunque si nos vamos a la documentación oficial, nos dice que la forma correcta es:

```
ArrayList <Object> nombreArray = new ArrayList <> ();
```

Es decir, entre ángulos se especifica y limita los objetos que puede contener el ArrayList. Si nuestro ArrayList va a contener objetos de la clase Producto, la forma adecuada de definir el ArrayList será

```
ArrayList <Producto> listaProductos = new ArrayList <> ();
```

Clases envoltorio o wrapper

El ArrayList sólo puede contener objetos, por lo que no puede contener datos de tipo primitivo (int, char, double, etc.) pero si necesitamos que nuestra lista contenga este tipo de datos lo vamos a hacer a través de las clases envoltorio que nos proporciona Java. Vamos a tener una clase equivalente a cada uno de los tipos básicos. También nos proporcionan métodos extras sobre los tipos de datos.

Primitiva	Wrapper	Constructor
boolean	Boolean	boolean o String
byte	Byte	byte o String
char	Char	char o String
int	Integer	int o String

Primitiva	Wrapper	Constructor
float	Float	float o String
double	Double	double o String
long	Long	long o String
short	Short	short o String

```
ArrayList<Integer> listaNumeros=new ArrayList<>();
```

Recorrido

Si tratamos con ArrayList parametrizados con una clase podemos parametrizar el Iterator o el bucle foreach, si no tendremos que hacer un cast al tipo de clase adecuado.


```
ArrayList<Persona> listaPersonas=new ArrayList<Persona>();
Iterator<Persona> it=listaPersonas.iterator();
while (it.hasNext()) {
    Persona persona=it.next();
}
```

o

```
ArrayList listaPersonas=new ArrayList();
Iterator it=listaPersonas.iterator();
while (it.hasNext()) {
    Persona persona=(Persona) it.next();
}
```

Copia de un ArrayList

Recordamos que un ArrayList es una clase. Para copiar un objeto no podemos hacer simplemente una asignación. Si lo hacemos así estaremos haciendo que las dos variables o atributos apunten a la misma referencia de memoria.

Para copiar un ArrayList tendremos que hacerlo a mano, copiando elemento a elemento, o podemos hacerlo pasando la referencia del array que queremos copiar al constructor de un nuevo ArrayList.

```
ArrayList<Objetos> nuevaCopia=new ArrayList<>(arrayACopiar);
```

ArrayList Bidimensionales

Podemos **simular** un Array bidimensional, los elementos del ArrayList serán ArrayList

```
ArrayList<ArrayList<Integer>> simularMatriz=new ArrayList();
simularMatriz.add(new ArrayList<>());
```

para modificar o acceder a los elementos.

```
simularMatriz.get(fila).add(valor);
simularMatriz.get(fila).get(columna);
```

Clase Collections

Esta clase contiene un conjunto de métodos estáticos que permiten operar sobre distintas colecciones como pueden ser los ArrayList que pueden ser muy útiles y que los usaremos frecuentemente.

- Collections.max(arraylist1) Devuelve el máximo elemento de arraylist1, de acuerdo al orden natural de sus elementos. También disponemos del análogo min(arraylist1).

- `Collections.reverse(arraylist1)` Invierte todos los elementos del `ArrayList`. No devuelve nada.
- `Collections.shuffle(arraylist1)` Intercambia aleatoriamente todos los elementos del `ArrayList`. No devuelve nada.
- `Collections.frequency(arraylist1, obj)` Devuelve el número de veces que aparece el objeto `obj` en `arraylist1`.
- `Collections.binarySearch(arraylist1,obj)` Busca el objeto `obj` en `arraylist1` y devuelve posición. Si no encuentra, devuelve `< 0`. El `ArrayList` debe estar ordenado.
- `Collections.sort(arraylist1)` ordena `arraylist1` ascendentemente. También disponemos del método `reverseOrder()` para ordenar descendentemente.

Puedes ver la lista completa en: <https://docs.oracle.com/javase/7/docs/api/java/util/Collections.html>

11.9 Ejercicios

a) Muestra el contenido del `ArrayList` después de ejecutar este código:

```
ArrayList <Long> a = new ArrayList <>();
for (int i=0;i<=9;i++)
    a.add((long)Math.pow(2,i+1));
```

b) Si sobre el `ArrayList` anterior ejecutamos el siguiente código ¿Cómo quedaría el `ArrayList`?

```
for (int i=0;i<a.size();i+=2)
    a.set(i,999);
```

c) Inventa dos ejemplos de lo que podría ser la salida de este código:

```
ArrayList <Integer> a = new ArrayList <>();
int lon=(int) (Math.random()*6)+5;
for (int i=0;i<lon;i++)
    a.add((int)(Math.random()*100)+1);
Collections.sort(a);
for (int i=0; i<a.size();i++)
    System.out.println(a.get(i));
```

d) ¿Es correcto este código? ¿Hace algo?

```
static void fun (ArrayList <String> a, String x) {while (a.remove(x)){{}}
```